NTCIP implementation of PMPP in CHART2

A discussion of the CHART2 implementation of the STMF and PMPP Protocols.

Edwards and Kelcey Technical Design Document for CHART2.

The PMPP Protocol as defined in the RFC 1101 draft of 1998, is a subset of the HDLC protocol. The PMPP protocol places constraints on the HDLC protocol in terms of the number of octets in fields as well as the structure of the INFO component. The PMPP protocol currently supports SNMP as the communications carrier for the frame. The STMF protocol is still in draft.

The following discussion covers the implementation of the PMPP Protocol in the NTCIP CHART2 Driver protocol. Covers issues like Java's handling of primitives, methods to achieve compliance with the specification, the design of the objects, integration with Opensource package joeSNMP and the integration into the CHART2 system.

Author	Cameron Riley
Email	criley@ekmail.com
Company	Edwards and Kelcey Technology, Inc
URL	http://www.ekcorp.com/
Address	750 Miller Drive, Suite F-1
City	Leesburg, Virginia
Zip	20175
phone	703.779.7988
fax	703.779.7989
date	5th December 2001

Abstract

The Point to Multi-Point Protocol is the NTCIP standard for communication with Dynamic Message Signs. PMPP is a specialization of HDLC which uses an SNMP PDU in the INFO field. The PMPP protocol also includes transparency requiring the bytes with value of 0x7E and 0x7D to be escaped. For packet integrity the protocol contains a Checksum field which uses the CCITT CRC-16 algorithm.

All primitives in Java are signed, this requires care to translate where necessary. The HDLC to PMPP hierarchy is represented in the source by the means of an HdlcEncoding object which contains methods to do the transparency. The working object for the PMPP package is the PmppRequest which manages the marshalling and unmarshalling of the binary PMPP requests and responses. The Facade for the package is the PmppBuilder which offers four simple methods.

The PMPP protocol uses SNMP for the data encoding. The library chosen to provide the PDU and BER Encoding was the joeSNMP library which is licensed under the LGPL. The CHART2 Facade to the joeSNMP package is the SnmpPdu object.

CHART

CHART is the highway incident management program of the Maryland State Highway Administration. CHART is comprised of four major components, traffic monitoring, incident response, traveler information and traffic management. The traveler information component of CHART provides real-time information concerning travel conditions on main roads in the primary coverage area. The information is conveyed to travelers via either Variable Message Signs, Highway Advisory Radio, Commercial radio and television broadcasts and a telephone advisory service. The variable message signs are programmable message boards, both permanent and portable which are capable of displaying real-time traffic conditions to motorists. NTCIP compliant Dynamic Message Signs are a type of VMS. CHART2 is the current implementation of the CHART system being developed by the Maryland State Highways Administration.

NTCIP

The National Transportation Communications for Intelligent Transportation System Protocol (NTCIP) is a family of standards maintained by NEMA, AASHTO and ITE. The NTCIP standards provide the rules and vocabulary for electronic traffic equipment from different manufacturers to communicate and operate with each other. NTCIP compliant devices must follow this standard. For a Device to be NTCIP compliant it must implement a mandatory set of MIB's, accept data transport by PMPP and be capable of reading Multi Message Format for sign display.

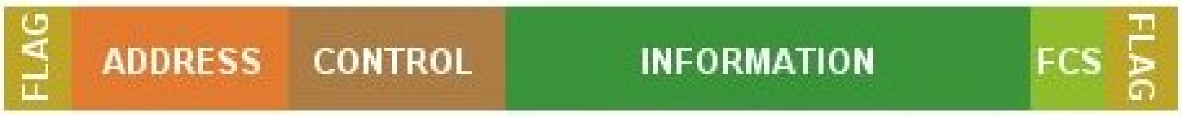
STMF

The Simple Transportation Management Framework is a set of standards that are part of the NTCIP specification which defines the manner in which information is processed, organized and exchanged amongst devices and stations. The Framework integrates the existing internet protocol, Simple Network Management Protocol or SNMP and it's ITS specific derivative the Simple Transportation Management Protocol or STMP.

HDLC Protocol

The High Level Data Link Control is the protocol for Frames. There are many variations of the HDLC protocol. PMPP subset of HDLC uses the asynchronous UCC or Unbalanced Connectionless Class of procedures which are defined in ISO/IEC 7809. The HDLC frame structure for PMPP is defined in the ISO/IEC 3309.

HDLC Frame Structure



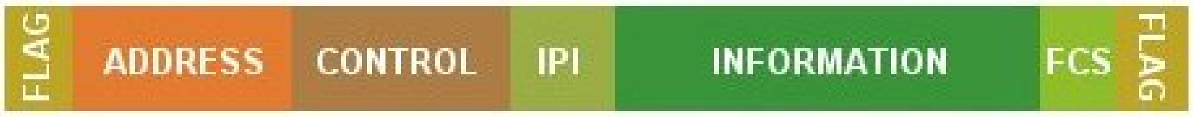
The structure of the HDLC protocol is defined by Frames that sit as bookends on the Frame. The HDLC FLAG is defined by the 8 bits, 01111110 which is represented in hex as 0x7E. The Address represents the address of the receiving device as an octet. The control field is an octet. The FCS is discussed in more detail later in the document and represents the Checksum on the fields in the frame.

Due to the framing of the HDLC protocol, the appearance of any 0x7E bytes in the binary fields would disrupt where the frames are parsed. For this reason the occurrence of any octets with the value 0x7E need to undergo transparency. The HDLC specification requires that any occurrences of 0x7E be escaped with the octet pair 0x7D 0x5E. The occurrences of the octet 0x7D is to escaped with 0x7D 0x5D.

PMPP Protocol

The Point to Multi Point Protocol a subset of the HDLC protocol and represents a means via the NTCIP standard to address the need for a specific Data Link Layer protocol which is written to the requirements of field devices such as Dynamic Message Signs (DMS).

PMPP Frame Structure



PMPP places restrictions on the HDLC protocol. The FLAG is as per the HDLC protocol.The ADDRESS field by PMPP can be a single octet or an extended value. The specification requires that the ADDRESS be no greater than 2 bytes and can incorporate group addressing. The address 0 and 63 are both reserved, 0 is never assigned and 63 is the address of the Primary Station. By the PMPP specification support for single byte addressing is mandatory, recognition of extended addressing is mandatory and support for the byte addressing is provisional. The CONTROL field is always a single octet.

The INFO field in PMPP is composed of the Initial Protocol Identifier(IPI) and the Data. For PMPP using the SNMP protocol, the DATA is represented by an SNMP PDU. The IPI field can be a single byte or an extended field following the same rules as the ADDRESS field, which can be found in OSI 3309. The IPI identifies the Protocol Data Unit that is encapsulated in the DATA field. For example, STMF is recognized by the IPI, as an unsigned byte of value 0xC1.

The extended algorithm is shown below. The least significant bit (lsb) is the bit on the far right and is the value for the 2^0 (or 2 to the power of zero). For extended field size such as addressing and IPI under the PMPP protocol. The first byte if it is not extended will have the lsb as 1. This can be AND'ed to test if the field is extended or not.

```
1101 1101
0000 0001 &
---------
0000 0001
```

In Java this can is represented in byte notation as byte&0x01

```
byte_value & 0x01
```

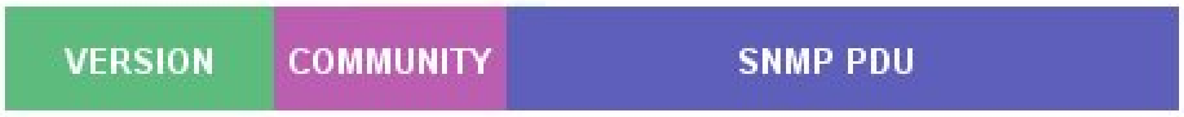
A result of 1 means it is a single byte field, a result of 0 indicates the field is an extended field. In both the cases of the Address and IPI, the recognition of extended fields is mandatory.

PMPP supports the Frame Types, Unnumbered Information (UI) and Unnumbered Poll (UP) which are defined in the ISO/IEC 7809 with UCC.

SNMP Protocol

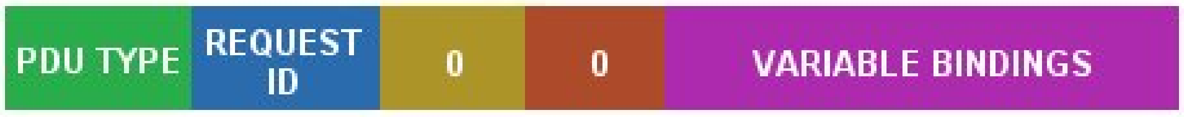
SNMP is the Simple Network Management Protocol, a commonly used simple protocol for communicating with remote devices to query and manipulate the Management Information Base or MIB's. The SNMP data unit is made up of the wrapper and the PDU or protocol data unit. The wrapper contains bytes describing the SNMP version and the Community ID.

SNMP Message Structure

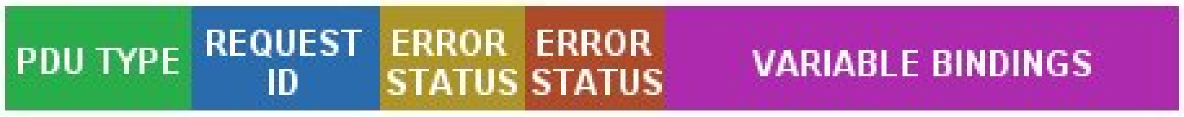


The bytes in the PDU differs slightly between the request and the response packets. A request packet comprises, the PDU type, which is the type of PDU being transmitted. For example a GetRequest PDU is represented as 0xA0 in hex and a Response PDU is represented by an 0xA2 identifier. The second field is the Request ID, this associates the requests with responses. The third and fourth fields are the Error Status fields. The final field is the Variable Bindings, these are stored as name value pairs. The PDU is commonly encoded with BER or Basic Encoding Rules.

SNMP Get, GetNext, Set, SNMPv2 Trap, Inform, Report PDU structure



SNMP Response PDU structure



Java SNMP Library

There are several Java SNMP libraries available both commercial and opensource, that were evaluated by Edwards and Kelcey Technology. The libraries were evaluated on their ability to fit in with the CHART2 DataPort interface, their developer seat cost and their runtime license cost. All were designed around the paradigm of an SNMP Session, in the case of CHART2 this is the role of the DataPort. Of more importance is that the library be capable of dumping the Snmp PDU as a byte array.

Three SNMP libraries were evaluated, the Adventnet SNMP API, the Outback Inc's SNMP Toolkit and the joeSNMP library. The Adventnet library had excorbant cost, the developer seat license being $11,500 alone. The Outback library was cheaper being $995 for the library and the deployment license. Both however were closed source. The joeSNMP library is licensed under the LGPL which is discussed in more detail in Appendix 2. The joeSNMP library has it's source publicly available and has a string use of byte buffers. The components of the SNMP PDU follow the SnmpSyntax interface which includes the encodeASN() and decodeASN() methods. As joeSNMP was the lowest on cost, the highest on soure availability and had the best API for retrieving and manipulating PDU's as byte array's, joeSNMP was chosen as the Java SNMP library which would best fit the project.

PMPP Problem Domain

Issue's surrounding PMPP implementation in CHART2;

- Java's use of unsigned and signed primitives.
- Adhering to the NTCIP Specification.
- Understanding the relationship between Pmpp, Hdlc and Snmp.

Java's 8 bit and 16 bit primitives

The Java language's primitives are of set lengths that aren't reliant on the platform they are compiled on, unlike C. The boolean primitive is 1 bit, the byte 4 bits, the short 8 bits, the char primitive 16 bits and the int 32 bits. On the JVM all primitives are stored as 32 bit, but padded so that it is transparent to the compiled application.

Unlike languages such as C, Java byte primitives are all signed. There is no unsigned keyword so int values of greater than 127 need to be subtracted from 256 and negated. The resulting octet has the same bit structure but fits into a Java signed byte. The Java equivalent of an unsigned byte is the char primitive.

```
/*
 * <p>For converting bytes that are unsigned and greater than 127
 * to a signed byte in the range -128 to 127. If the int is less
 * than or equal to 127 it can be represented as a signed byte
 * by casting.
 * @param i, the int to be converted to a signed byte
 * @return byte, the signed byte
 */
public byte getSignedByte(int i)
{
    if(i<+127)
    {
        return (byte)i;
    }

    /*
```

```
         * Unsigned byte to signed
         */
        return (byte)(-(256 - i));
    }
```

Adhering to the NTCIP Specification

Description of the RFC's and where the data is that is most important towards a correct implementation of the PMPP protocol.

NTCIP RFC 2301 specifies the standards for the Simple Transportation Management Framework. The RFC covers the Transportation layer relationships, the SNMP requirements, the SMI requirements, the MIB groups and the STMP requirements. Section 2 of RFC 2301 describes the requirements for conformance to the STMF Standard.

Section 2.2.5.4 of RFC 2301 allows for direct SNMP messages directly by a sub-network interface. In the case of Dynamic Messaging Signs, there is need for the signs address to sent in the packet. PMPP satisfies this requirement.

The STMF standard places no demands on the SNMP message types beyond SNMPv1. The message types that are required to be supported by vendors are GetRequest, GetNextRequest, GetResponse, SetRequest or Trap PDU's. As STMP, which is an extended version of SNMP for transportation devices is still in draft, it will not be studied further for this document.

NTCIP RFC 2101 specifies the details of the PMPP protocol and it's usage as an HDLC specialization, along with SNMP. The document contains details on the restrictions on the HDLC fields as well as the extra field information the PMPP protocol carries.

The Relationship between HDLC, PMPP and SNMP

The PMPP protocol is a specialization of the HDLC protocol, in particular the HDLC Unbalanced Connectionless Class which is defined in ISO/IEC 7809. PMPP places restrictions on HDLC field sizes plus adds an extra field, the IPI in the INFO field. The transmission of the frame is in the start/stop mode with basic transparency. This is the escaping for the occurrence of 0x7E and 0x7D. PMPP supports extended fields in the same manner as HDLC. PMPP uses the 16-bit FCS which is described in ISO/IEC 3309.

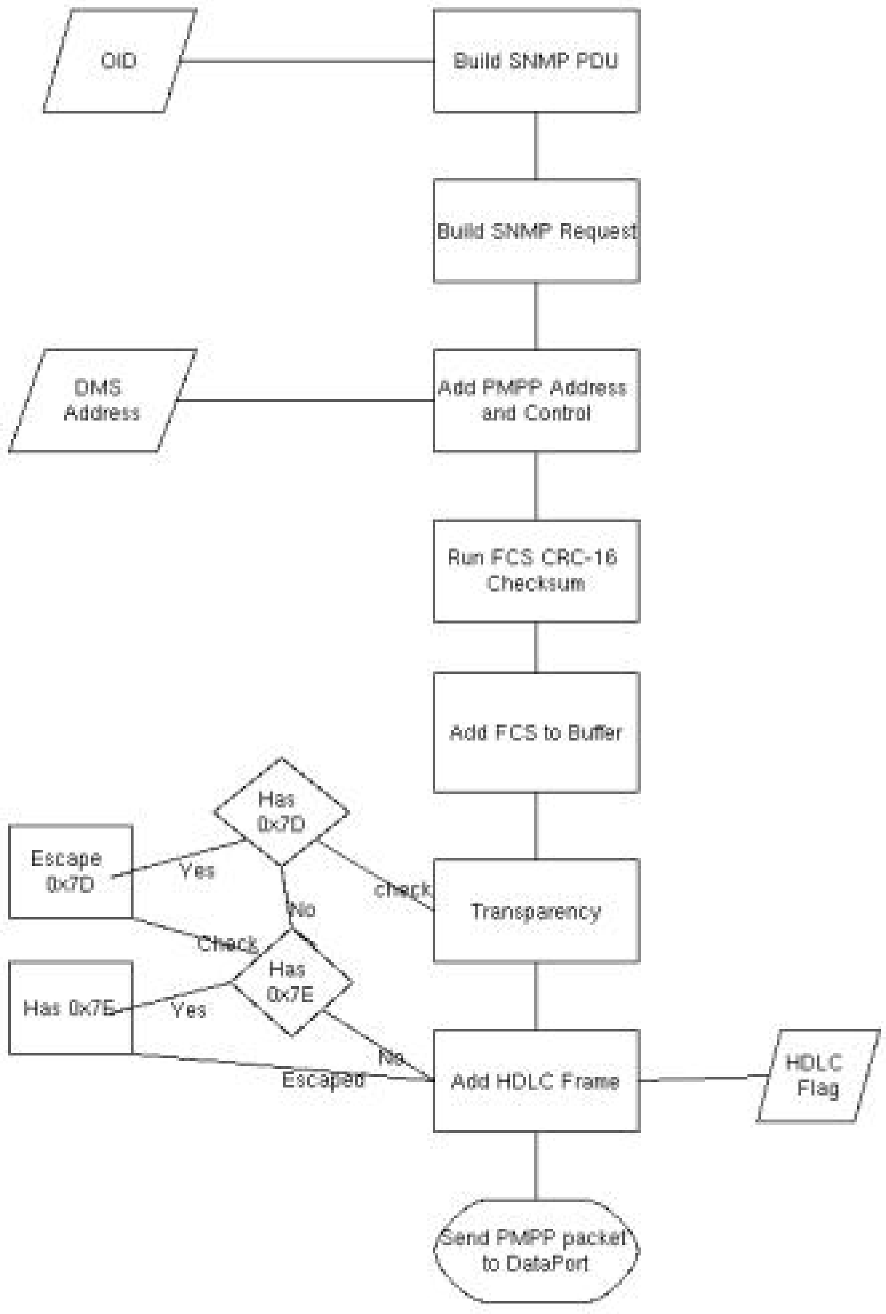
Encoding PMPP Packets

HDLC Frames undergo transparency during their encoding. As the HDLC frames are have as their delimiter the 0x7E flag, the packet is escaped or encoded with the following;

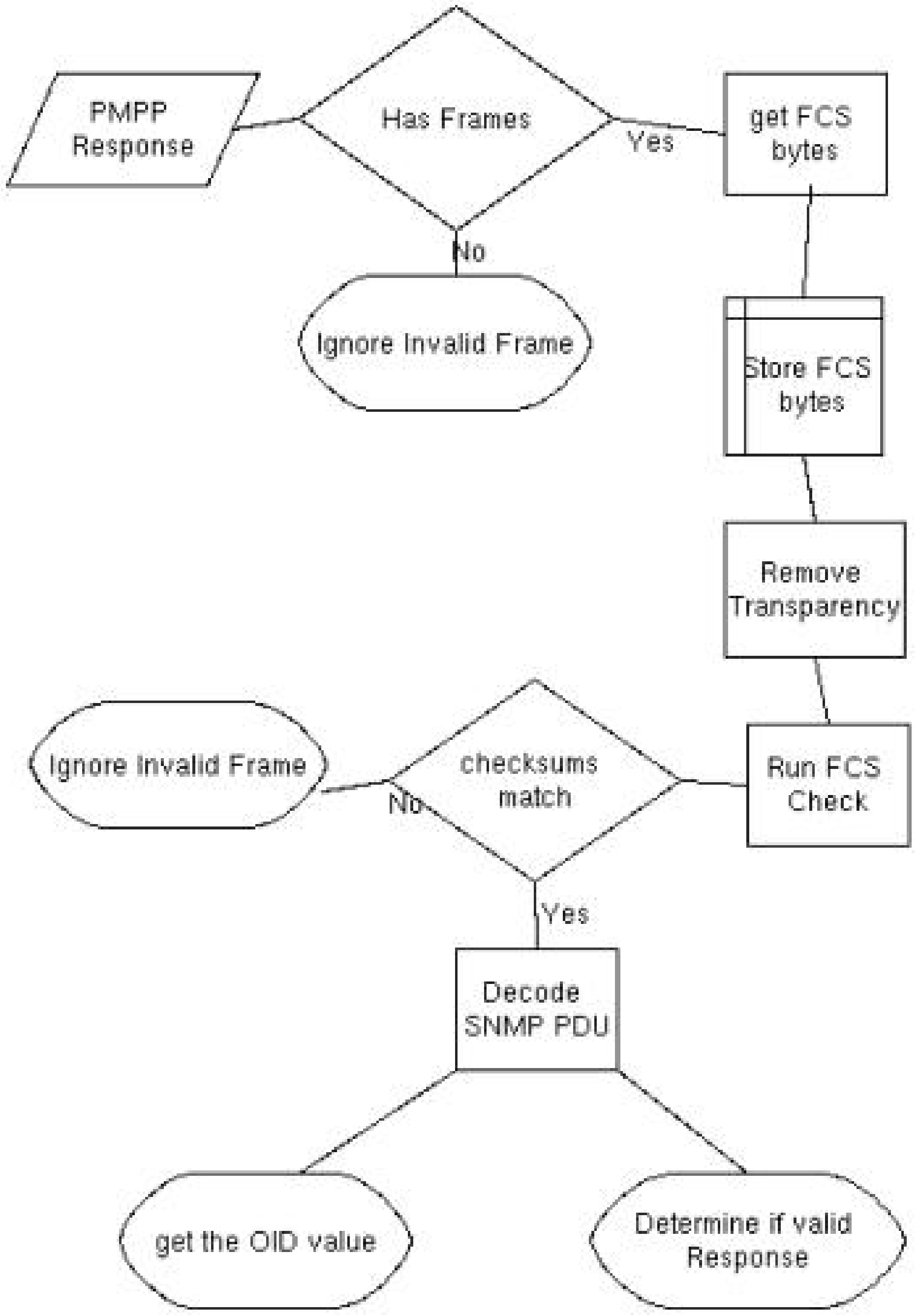
```
0x7e is encoded as 0x7d, 0x5e.    (Flag Sequence)
0x7d is encoded as 0x7d, 0x5d.    (Control Escape)
```

The specification requires that the checksum be calculated before the processing of transparency, before the addition of the FCS and the Frames but after the addition of the Address and Control fields.

PMPP Encoding Flow



PMPP Decoding Flow



FCS

The FCS field is a 16 bit field which contains the checksum for the frame. The checksum is calculated over all bits of the Address, Control, Protocol, Information and Padding fields. This doesn't include for any bits or octets inserted for transparency. This also does not include any flag sequences or the FCS field itself.

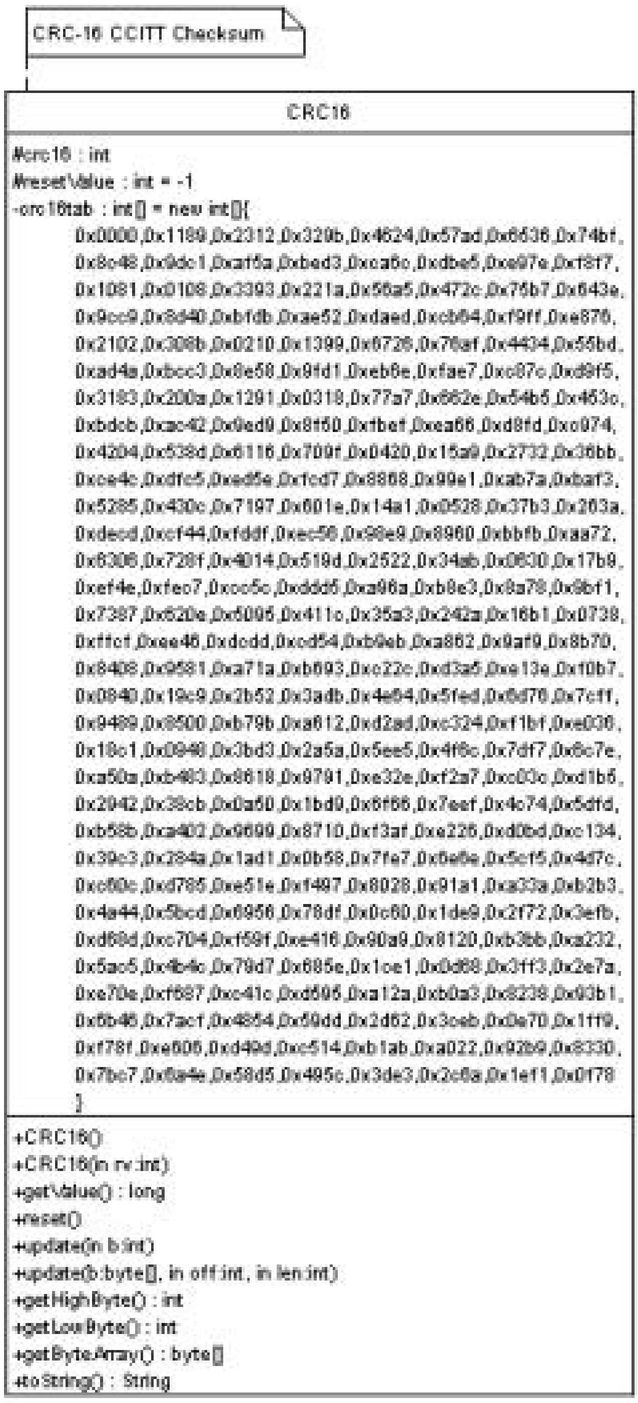
The algorithm for the 16 bit CRC is contained in ISO/IEC 3309 section 4.6.2 and is the remainder of the division (modulo 2) by the generator polynomial;

```
x^16 + x^12 + x^5 + 1
```

where in the line above, the character ^ represents to 'the power of'. Using a look-up table the algorithm appears as;

```
crc16 = (crc16 >> 8) ^ crc16tab[(crc16 ^ off++) & 0xff];
```

where crc16 is the int representing the checksum, crc16tab is the pre-calculated table and off is the offset of the byte[] of the frame being checksummed.



```
CRC-16 CCITT Checksum

CRC16
#crc16 : int
#resetValue : int = -1
-crc16tab : int[] = new int[]{
        0x0000,0x1189,0x2312,0x329b,0x4624,0x57ad,0x6536,0x74bf,
        0x8c48,0x9dc1,0xaf5a,0xbed3,0xca6c,0xdbe5,0xe97e,0xf8f7,
        0x1081,0x0108,0x3393,0x221a,0x56a5,0x472c,0x75b7,0x643e,
        0x9cc9,0x8d40,0xbfdb,0xae52,0xdaed,0xcb64,0xf9ff,0xe876,
        0x2102,0x308b,0x0210,0x1399,0x6726,0x76af,0x4434,0x55bd,
        0xad4a,0xbcc3,0x8e58,0x9fd1,0xeb6e,0xfae7,0xc87c,0xd9f5,
        0x3183,0x200a,0x1291,0x0318,0x77a7,0x662e,0x54b5,0x453c,
        0xbdcb,0xac42,0x9ed9,0x8f50,0xfbef,0xea66,0xd8fd,0xc974,
        0x4204,0x538d,0x6116,0x709f,0x0420,0x15a9,0x2732,0x36bb,
        0xce4c,0xdfc5,0xed5e,0xfcd7,0x8868,0x99e1,0xab7a,0xbaf3,
        0x5285,0x430c,0x7197,0x601e,0x14a1,0x0528,0x37b3,0x263a,
        0xdecd,0xcf44,0xfddf,0xec56,0x98e9,0x8960,0xbbfb,0xaa72,
        0x6306,0x728f,0x4014,0x519d,0x2522,0x34ab,0x0630,0x17b9,
        0xef4e,0xfec7,0xcc5c,0xddd5,0xa96a,0xb8e3,0x8a78,0x9bf1,
        0x7387,0x620e,0x5095,0x411c,0x35a3,0x242a,0x16b1,0x0738,
        0xffcf,0xee46,0xdcdd,0xcd54,0xb9eb,0xa862,0x9af9,0x8b70,
        0x8408,0x9581,0xa71a,0xb693,0xc22c,0xd3a5,0xe13e,0xf0b7,
        0x0840,0x19c9,0x2b52,0x3adb,0x4e64,0x5fed,0x6d76,0x7cff,
        0x9489,0x8500,0xb79b,0xa612,0xd2ad,0xc324,0xf1bf,0xe036,
        0x18c1,0x0948,0x3bd3,0x2a5a,0x5ee5,0x4f6c,0x7df7,0x6c7e,
        0xa50a,0xb483,0x8618,0x9791,0xe32e,0xf2a7,0xc03c,0xd1b5,
        0x2942,0x38cb,0x0a50,0x1bd9,0x6f66,0x7eef,0x4c74,0x5dfd,
        0xb58b,0xa402,0x9699,0x8710,0xf3af,0xe226,0xd0bd,0xc134,
        0x39c3,0x284a,0x1ad1,0x0b58,0x7fe7,0x6e6e,0x5cf5,0x4d7c,
        0xc60c,0xd785,0xe51e,0xf497,0x8028,0x91a1,0xa33a,0xb2b3,
        0x4a44,0x5bcd,0x6956,0x78df,0x0c60,0x1de9,0x2f72,0x3efb,
        0xd68d,0xc704,0xf59f,0xe416,0x90a9,0x8120,0xb3bb,0xa232,
        0x5ac5,0x4b4c,0x79d7,0x685e,0x1ce1,0x0d68,0x3ff3,0x2e7a,
        0xe70e,0xf687,0xc41c,0xd595,0xa12a,0xb0a3,0x8238,0x93b1,
        0x6b46,0x7acf,0x4854,0x59dd,0x2d62,0x3ceb,0x0e70,0x1ff9,
        0xf78f,0xe606,0xd49d,0xc514,0xb1ab,0xa022,0x92b9,0x8330,
        0x7bc7,0x6a4e,0x58d5,0x495c,0x3de3,0x2c6a,0x1ef1,0x0f78
        }
+CRC16()
+CRC16(in rv:int)
+getValue() : long
+reset()
+update(in b:int)
+update(b:byte[], in off:int, in len:int)
+getHighByte() : int
+getLowByte() : int
+getByteArray() : byte[]
+toString() : String
```

Utility Package UML

The manipulation of the PMPP packet is done with byte array's in the software. For this reason, the package required a set of methods to manipulate byte array's. Methods such as add a byte, remove a byte, set a byte and frame an array set.

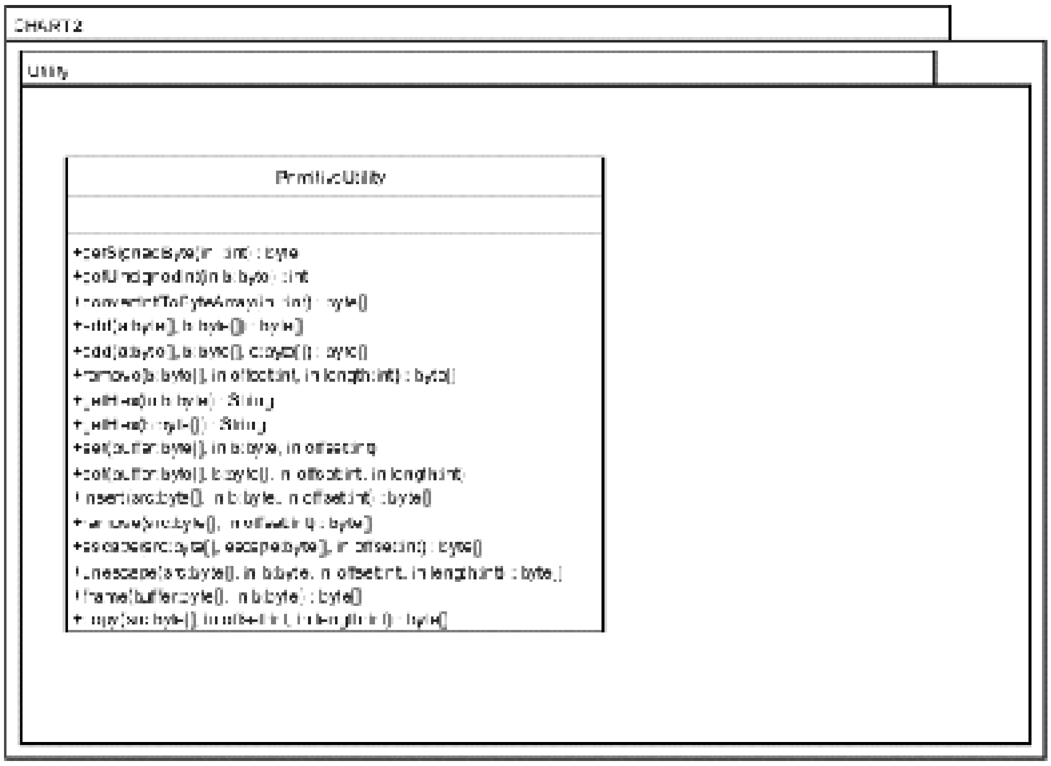


CHART2
Utility
PrimitiveUtility
+getSignedByte(in i:int) : byte
+getUnsignedInt(in b:byte) : int
+convertIntToByteArray(in i:int) : byte[]
+add(a:byte[], b:byte[]) : byte[]
+add(a:byte[], b:byte[], c:byte[]) : byte[]
+remove(b:byte[], in offset:int, in length:int) : byte[]
+getHex(in b:byte) : String
+getHex(b:byte[]) : String
+set(buffer:byte[], in b:byte, in offset:int)
+set(buffer:byte[], b:byte[], in offset:int, in length:int)
+insert(src:byte[], in b:byte, in offset:int) : byte[]
+remove(src:byte[], in offset:int) : byte[]
+escape(src:byte[], escape:byte[], in offset:int) : byte[]
+unescape(src:byte[], in b:byte, in offset:int, in length:int) : byte[]
+frame(buffer:byte[], in b:byte) : byte[]
+copy(src:byte[], in offset:int, in length:int) : byte[]

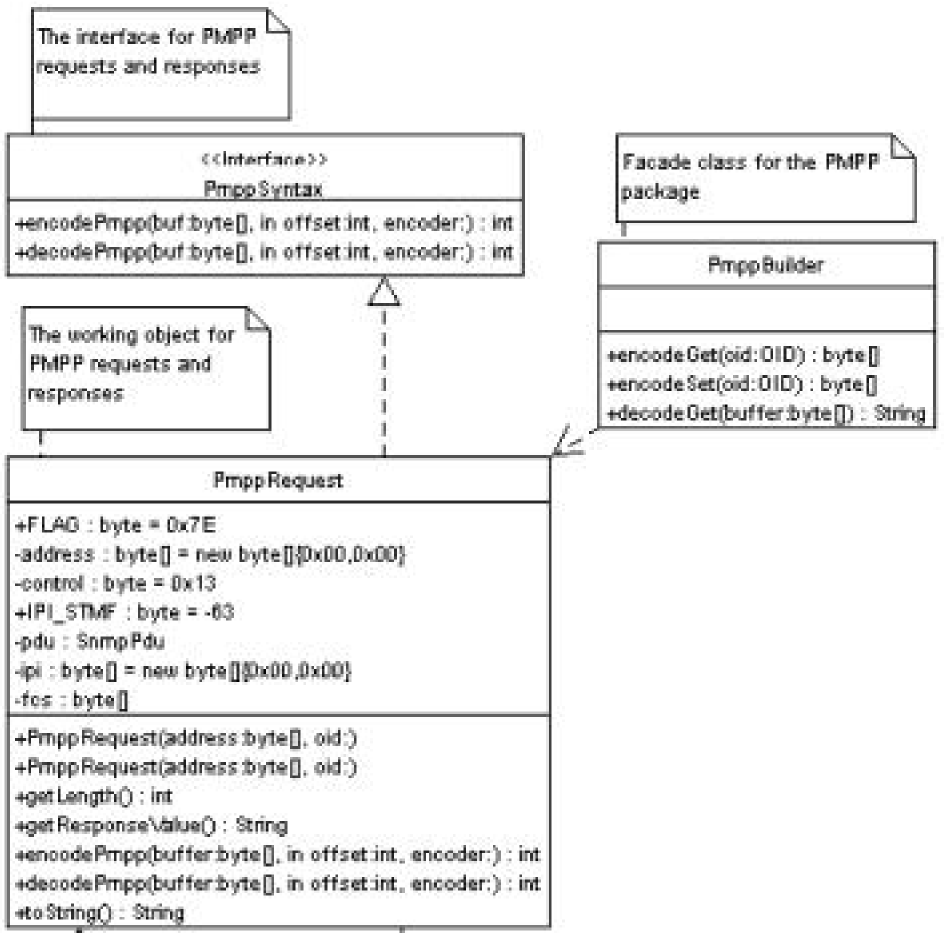
PMPP Package UML

The package is exposed to the CHART2 system via the PmppBuilder object which follows the Facade pattern. The PmppBuilder exposes public method to decode and encode SNMP Get and Set Requests. The decoding method for set checks that no error codes were returned with the packet.

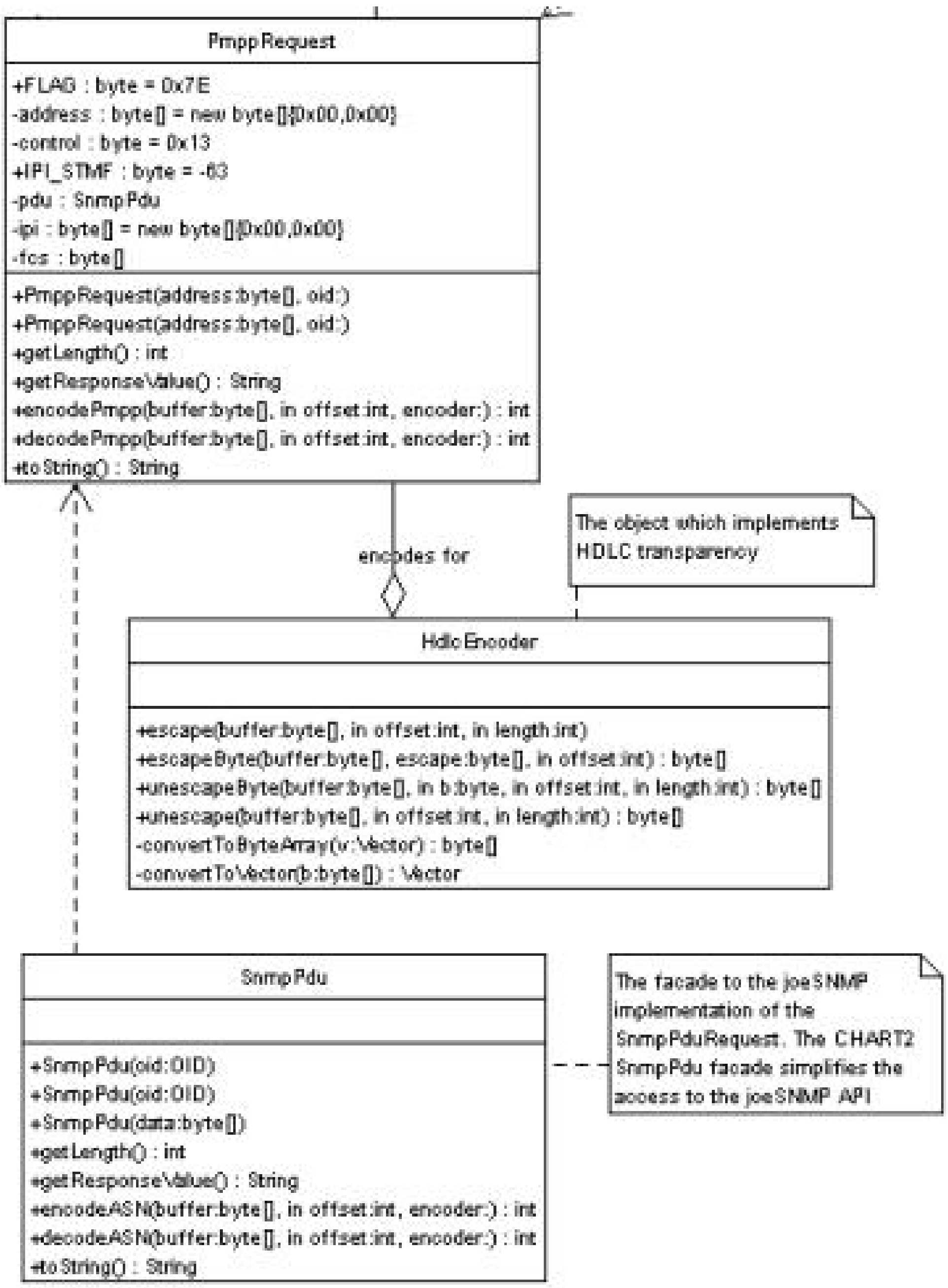
The working object is the PmppRequest, this object controls the marshalling and unmarshalling of the Pmpp frames to their fields which apart from the SnmpPdu, are stored in the Request class as class member variables. For unmarshalling, the PDU's returned value for the OID query is the only information of interest other than the error codes. Through chain of responsibility the getRequestValue() method is publicly exposed for the PmppBuilder to access.

The SnmpPdu serves as the facade to the joeSNMP library. The joeSNMP contains the SnmpPduRequest object which the CHART2 SnmpPdu requests for the BerEncoded Pdu. The PmppRequest Object uses the HdlcEncoder object to run the transparency through the PMPP buffer.

PMPP Request and PMPP Syntax Interface Interaction


The interface for PMPP requests and responses
<<Interface>>
PmppSyntax
+encodePmpp(buf:byte[], in offset:int, encoder:) : int
+decodePmpp(buf:byte[], in offset:int, encoder:) : int
Facade class for the PMPP package
PmppBuilder
+encodeGet(oid:OID) : byte[]
+encodeSet(oid:OID) : byte[]
+decodeGet(buffer:byte[]) : String
The working object for PMPP requests and responses
PmppRequest
+FLAG : byte = 0x7E
-address : byte[] = new byte[]{0x00,0x00}
-control : byte = 0x13
+IPI_STMF : byte = -63
-pdu : SnmpPdu
-ipi : byte[] = new byte[]{0x00,0x00}
-fcs : byte[]
+PmppRequest(address:byte[], oid:)
+PmppRequest(address:byte[], oid:)
+getLength() : int
+getResponseValue() : String
+encodePmpp(buffer:byte[], in offset:int, encoder:) : int
+decodePmpp(buffer:byte[], in offset:int, encoder:) : int
+toString() : String

PMPP Request and SnmpPdu Interaction


PmppRequest
+FLAG : byte = 0x7E
-address : byte[] = new byte[]{0x00,0x00}
-control : byte = 0x13
+IPI_STMF : byte = -63
-pdu : SnmpPdu
-ipi : byte[] = new byte[]{0x00,0x00}
-fcs : byte[]
+PmppRequest(address:byte[], oid:)
+PmppRequest(address:byte[], oid:)
+getLength() : int
+getResponseValue() : String
+encodePmpp(buffer:byte[], in offset:int, encoder:) : int
+decodePmpp(buffer:byte[], in offset:int, encoder:) : int
+toString() : String
encodes for
The object which implements HDLC transparency
HdlcEncoder
+escape(buffer:byte[], in offset:int, in length:int)
+escapeByte(buffer:byte[], escape:byte[], in offset:int) : byte[]
+unescapeByte(buffer:byte[], in b:byte, in offset:int, in length:int) : byte[]
+unescape(buffer:byte[], in offset:int, in length:int) : byte[]
-convertToByteArray(v:Vector) : byte[]
-convertToVector(b:byte[]) : Vector
SnmpPdu
+SnmpPdu(oid:OID)
+SnmpPdu(oid:OID)
+SnmpPdu(data:byte[])
+getLength() : int
+getResponseValue() : String
+encodeASN(buffer:byte[], in offset:int, encoder:) : int
+decodeASN(buffer:byte[], in offset:int, encoder:) : int
+toString() : String
The facade to the joeSNMP implementation of the SnmpPduRequest. The CHART2 SnmpPdu facade simplifies the access to the joeSNMP API

Appendix 1 : The benefits of Opensource Libraries

Opensource projects and licenses are more prevalent today than they were 5 years ago, much of this is due to the media time Linux received during the technology boom of the 1990's. Opensource's value is it's code, it allows software projects to re-use existing, and more importantly tested code. An opensource project is defined by it's publicly published software code. The benefits are that the code is freely available for public use and more importantly for public contribution. This allows anyone to contribute to the project and advance the software codebase. This gives great empowerment to the developer and a large codebase of code to use in projects. With this comes the responsibility both legal and social to adhere to the license the code is published under. In some cases there are restrictions placed on how a derived work may be used, this is discussed further in the Appendix 2.

Edwards and Kelcey Technology has used opensource libraries in previous projects with great success. This has the benefit of our software engineering group not to have to re-invent the wheel each time to satisfy the requirements of a project. Several of our projects have used XML libraries, in particular the FMSuite series of products. As the focus of the FMSuite project was to solve operational problems with managing Facilities, by using the Apache XML libraries we were able to focus on the Facility rather than being sidetracked by having to solve low-level software engineering problems first. The other direct benefit is the capability of contributing to the project to advance it for our particular needs. This occurred with the Apache Turbine project.

Apart from the obvious saving's in cost by not purchasing an equivalent commercial library or software product, as an engineering company, the access to the source is empowering. It means we are not at the vendors beck and whim to add additional functionality. With an opensource project we are able to add the additional functionality as it is needed and then publish it back to the project for peer review.

Appendix 2 : Differences between Opensource Licenses

The main opensource licenses are the BSD license, the Apache Public License, the GNU Public License also known as the GPL, the Lesser GPL better known as the LGPL or the Library GPL and the Mozilla Public License commonly known as the MPL. All satisfy the opensource description defined by the opensource definition which can be found at www.opensource.org. There are also many other licenses which satisfy the opensource definition.

All licenses despite allowing for opensource software to be distributed have differences in what constitutes a derived work. This is where the licenses place restrictions on the manner in which the source can be distributed. A derived work is any body of work which incorporates the source code carrying one of these opensource licenses. There are different ways in which the source code can be incorporated into the new project, such as direct embedding, package integration, library integration or compile time linking.

The GPL has the most restrictive of what constitutes a derived work, given it's origins in a C world, static linking with a GPL'ed project is seen as being a derived work. Derived work also includes a project which uses any GPL licensed source code in it. If the work is not distributed and only used privately then none of this is an issue. It is only when the work is distributed, that to be distributable, the projects code to be distributed alongside GPL code, must be distributed under the GPL as well. This is an aspect of using GPL software that must be born in mind. If this is unacceptable to the project, then GPL software cannot be used to create a derived work.

The LGPL allows for linking, importing and compiling against, to create a derived work without the projects source having to be licensed under the LGPL or GPL to be distributed. For this reason it is known as the Lesser or Library GPL. If however the project modifies the code in the LGPL'ed library and then distributes the library with the project, any modified code in the library must be published to whomever the project is distributed to. For most commercial concerns the LGPL is free enough and open enough without the GPL's restrictive clause.

The BSD license is the most liberal of licenses and used for the operating systems such as FreeBSD, OpenBSD and NetBSD. Other project which use the BSD license for their projects includes the X11 windowing system for Linux. The BSD license was also the basis for the Apache License. The BSD license states that the author of the code gives no warranty to it's function and that the authors copyright remain on the source. A BSD licensed project can be compiled to a binary without the derived work's source code having to be published to the customer or the public.

The Apache license is used for the well known Apache projects. Such as the Apache Webserver, Jakarta Tomcat, Jakarta Turbine, Xalan and Xerces XML projects. The Apache website is a wealthy resource for software engineers. The Apache license follows the BSD style of license.

The Mozilla Public License, or MPL grew out of the opensourcing of the Netscape codebase and was an attempt to balance opensource fundamentals with the wants and desires of for-profit business. A derived work with the MPL requires the modified code be published back to the project. The Mozilla Browser project is published under a dual MPL/GPL license and probably represents the most successful attempt at a commercial opensource project.

Appendix 3 : Edwards and Kelcey Commercial Support for Opensource Libraries

One of the recurring area's of concern for departments using opensource libraries or source code is the question of support. As opensource projects publish their code publicly, Edwards and Kelcey Technology can support any opensource source code used in any project. Edwards and Kelcey Technology can be contracted to do code maintenance on an opensource software project or add new functionality and features to the opensource software project. Edwards and Kelcey Technology has a great deal of experience supporting opensource libraries and projects.

Resources

- CHART2 : http://www.chart.state.md.us/
- Edwards and Kelcey Technology : http://www.ekcorp.com/
- NEMA : http://www.nema.org/
- NTCIP : http://www.ntcip.org/

References

- Edwards and Kelcey Report Subtask 1 : NTCIP Compliance Survey and Driver Development. 2001.
- Edwards and Kelcey Report Subtask 2 : NTCIP Driver High Level Design. 2001.
- RFC 1662 PPP in HDLC-like Framing : http://www.armware.dk/RFC/rfc/rfc1662.html
- MSHA Report : CHARTII Release I, Build 2A - High Level Design.

Glossary

- BER : Basic Encoding Rules.
- Bit : Binary Digit.
- DMS : Dynamic Message Sign.
- FMS : Field Management Station.
- HDLC : High-level Data Link Control.
- HSB : Highest Significant Bit.
- IANA : Internet Assigned Number Authority.
- ITS : Intelligent Transportation Systems.
- LSB : Lowest Significant Bit.
- MIB : Management Information Base.
- NEMA : National Electrical Manufacturers Association.
- NTCIP : National Transportation Communications for ITS Protocol.
- OID : Object Id.
- PDU : Protocol Data Unit.
- PMPP : Point to Multi Point Protocol.
- SNMP : Simple Network Management Protocol.
- UI : Unnumbered Information.
- UP : Unnumbered Poll.

- VMS : Variable Message Sign.
- WAN : Wide Area Network.